REMARKS/ARGUMENTS

Claims 1-19 are pending and rejected.

Claims 1-5, 8 and 11-14 are rejected under 35 U.S.C. § 102(b) as being taught by Peleg et al., (hereinafter "Peleg"), US Pat. No. 5,381,533. Claims 6-7 and 15-19 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Peleg, US Pat No. 5,381,533, and further in view of Rotenberg et al., (hereinafter "Rotenberg"), "A Trace Cache Microarchitecture and Evaluation " IEEE © 1999. Claims 9-10 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Peleg, US Pat. No. 5,381,533, and further in view of Kyker et al., (hereinafter "Kyker"), US Pat. No. 6,578,138.

Applicants submit the cited references do not teach, suggest or disclose at least "[a] cache comprising: a cache line to store an instruction segment further comprising a plurality of instructions stored in sequential positions of the cache line in reverse program order, wherein a conditional branch causing program flow to jump from a first location in a first instruction stream to a second location in a second instruction stream causes a terminal instruction from the second instruction stream to be stored in a first position of a cache line" (*e.g.*, as described in claim 1).

The Office Action asserts Peleg teaches the relevant limitations at column 1, line 64 to column 2, line 19, column 4, lines 11-23, column 8, line 30 – column 9, line 35, column 10, lines 1-24, and column 10, line 58 to column 11, line 12, and cites Figure 5A, 5B, and 6-12 generally. *See* Office Action dated 8/18/2009, paragraph 11. Applicants disagree.

The first cited section, column 1, line 64 to column 2, line 19, consists of the entire Summary of the Invention section. It generally describes that the cited reference is directed to

storing data in a cache memory. More specifically, it states it is directed to identifying trace segments of instructions in a computer program in the order they are executed. Applicants submit it does not describe the specific, relevant limitations of claim 1.

The second cited section, column 4, lines 11-24, is similarly general. It describes that a "basic block" as defined by the reference comprises instructions in a computer program which are unconditionally and consecutively executed and begins after execution of branch instruction and ends with the execution of another branch instruction. It further states that once a first instruction in a basic block is executed, all the remaining instruction in the basic block will for certain be executed since there are no instructions in the basic block that go beyond an inserted branch instruction. Applicants submit this cited section does not teach or suggest at least the relevant, specific limitations of claim 1 discussed above. For example, contrary to the Office Action's assertion, this section fails to teach or suggest at least "…wherein a conditional branch causing program flow to jump from a first location in a first instruction stream to a second location in a second instruction stream causes a terminal instruction from the second instruction stream to be stored in a first position of a cache line" (*e.g.*, as recited in claim 1). *See* Office Action dated 8/18/2009, paragraph 7.

Next, the Office Action cites to the extensive section of column 8, line 30 – column 9, line 35. First, Applicants submit that because of the extensive nature of the section, Applicants have no choice but to discuss the section in summary. Applicants respectfully request further clarification of the current rejection by citation to a more specific portion of the cited section.

Regardless, Applicants maintain the cited section fails to teach or suggest the relevant limitations. It is directed to the description of Figures 5A, 5B. Figure 5A describes a static

sequence of computer program instructions and the flow that may occur during execution of the instructions. FIG. 5B illustrates the order of execution of blocks of the instruction for the flow of FIG. 5A. The cited section sequentially describes in detail execution by the computer of program instruction according to the method described (*e.g.*, "…At the end of this block the branch is taken and BB.sub.N+1 is next executed (BB.sub.3). At the end of BB.sub.3 the branch is not taken and BB.sub.4 is executed…"). *See e.g.*, column 8, lines 30-43. Applicants submit that similar to the cited sections discussed above, this cited section fails to teach or suggest at least a conditional branch causing program flow to jump from a first location in a first instruction stream to a second location in a second instruction stream causes a terminal instruction from the second instruction stream to be stored in a first position of a cache line" (*e.g.*, as described in claim 1).

The Office Action also cites to column 10, lines 1-24 which fail to teach or suggest the relevant limitations as well. This cited section describes Figure 9, which, similar to the previous cited section describes the sequential execution of computer program instructions according to the described method. It describes, for example, storing blocks twice in the array. The reference states this assures that the CPU can access a block in cache even where the branching is different than predicted. It also describes specific steps as described in Figures 9 and 10 (*e.g.*, "As shown in FIG. 9, after BB.sub.4 is executed it is transferred into the line 75 in a position after BB.sub.3 and into the beginning of line 73. As shown in FIG. 10, BB.sub.3 and BB.sub.4 are transferred to the array along with the address A.sub.3 and the next address A.sub.12.

After execution of BB.sub.4, the CPU will request the instruction at A.sub.12. When it does, a hit will occur as shown in FIG. 10."). Again, Applicants submit this cited section fails to

teach or suggest the relevant limitations of claim 1 discussed above.

The last cited section, column 10, line 58 to column 11, line 12 describes generally the benefits from implementing the methods as described in the reference. In sum, it states that if multiple blocks are repeatedly executed (in a loop), the instructions in the loop will be continuously supplied to the CPU with only a single address being sent to the cache memory. However, this section fails to teach or suggest the specific limitations of claim 1 discussed above.

Rotenberg and Kyker fail to make up for the deficiencies of Peleg. Rotenberg is directed to increasing fetch bandwidth and decreasing performance bottlenecks. Kyker is directed to various methods of storage and unrolling of instruction loops. Applicants submit these references fail to make up for the deficiencies of Peleg for at least the reasons discussed in previous response, and hereby maintain and incorporate by reference all arguments made in previous responses.

Therefore, since for at least the above-discussed reasons, the cited references fail to teach or suggest each and every limitation of claim 1, they fail to support proper § 102 or § 103 rejections. Applicants submit claim 1 is allowable. Independent claims 5, 6, 8 and 14 contain similar allowable limitations, and are therefore allowable for similar reasons. Claims 2-4, 7, 9-13 and 15-19 are allowable for depending from allowable base claims.

For at least all the above reasons, the Applicant respectfully submits that this application is in condition for allowance. A Notice of Allowance is earnestly solicited.

The Examiner is invited to contact the undersigned at (408) 975-7500 to discuss any matter concerning this application.

The Office is hereby authorized to charge any additional fees or credit any overpayments under 37 C.F.R. §1.16 or §1.17 to Deposit Account No. **11-0600**.

Respectfully submitted,

KENYON & KENYON LLP

Date: October 19, 2009

By: /Sumit Bhattacharya/
Sumit Bhattacharya
(Reg. No. 51,469)
Attorneys for Intel Corporation

KENYON & KENYON LLP
333 West San Carlos St., Suite 600
San Jose, CA 95110

Telephone: (408) 975-7500
Facsimile: (408) 975-7501